Course reader: *Scripts and functions, part II*

- When you invoke a function, MATLAB creates a separate little miniverse for that function (or maybe a teenyverse for a function inside a function). (They are technically called *workspaces*, but I prefer *miniverses*.) Inside a function's miniverse, the variables in the main workspace or the workspace of any other function are not immediately accessible. This is why functions need inputs.

- Normally, MATLAB will create and destroy these miniverses when calling and completing a function. However, you can enter a functions miniverse by "stepping into" the function. This is done by clicking on the little horizontal line next to a line number. The command prompt will change to `K>>` (I think K stands for keyboard).

- To quit workspace and get back to the previous miniverse in the stack, type `dbquit` or click on the red button *Quit Debugging* from the *Editor* menu.

- Stepping into functions is mainly used to debug or work on functions, but it's also a great way to learn how that function works, which will make you a better programmer.

- It's also possible to have MATLAB bring you into a function's miniverse whenever there is an error or warning. This is done by clicking in the menu on *Editor* → *Breakpoints* → *Stop on Errors*

- A function's "help text" is what you see in the command window when you type `help functionName`. All functions need help text, even little internal functions. The help text should include:
  1. The name of the function and a list of input and output variable ordering.
  2. A description of the function.
  3. A detailed list of all mandatory and optional inputs.
  4. A detailed list of outputs.
  5. (optional) additional relevant information
  6. (optional) examples of this function
  7. (optional) other related functions
  8. Your name and/or contact information

- It may be tempting to write lots of functions to automate everything, but... just don't. Having too many functions causes confusion, can slow down your progress, and makes it annoying to share your code with other people. Create new functions only when they are necessary.

- **MOST IMPORTANT**: *Never change built-in functions or toolbox functions*. If you want to change the behavior of some function, best practice is to copy the function to a separate file and use a different function name. For example, you might have your own function `mymean` instead of changing the mean.m file.

Exercises

1. This exercise extends exercise 1 in the *Scripts and functions, part I* video. Add to this function by checking the input to make sure it's OK for the rest of the code. You should check for the following:
   1. `m` and `n` are integers between 1 and 30.
   2. `s` is greater than .1

2. Write a function that computes the arithmetic mean of a set of numbers (sum the numbers and divide by the number of numbers). Include a help text and the following checks on the input:
   1. The input must be numeric (not strings, characters, cells, etc).
   2. Only vector inputs are allowed (not matrices).
   3. There must be at least 5 elements.

3. Use the `mod` function to transform the vector 1:6 into 1-2-3-1-2-3. It's possible to do with one line and no control statements.

4. Make a function out of the previous exercise. There should be two inputs: `n` and `k`, where `n` is the number of integers (e.g., in the previous exercise, n=6) and `k` is the number of counting integers (k=3 in the previous exercise). Make sure your function works for different values of `n` and `k`. The function needs help text, and you should check the inputs (you can determine what the appropriate input checks are). The output of the function is the new vector.

5. You can access information across stacks using the functions `evalin` and `assignin`. Step into the `linspace` function and place the `c` variable into the main stack. (Reminder: don't make any changes to the function itself!)

6. The MATLAB profiler is a handy tool that will analyze a function and show you which lines take how long to run, which in turn helps you make your code more efficient. Put the code below into a function called `testfun.m`.

```
n = 4000;
v = zeros(n,1);
for i=1:n
    d = det(repmat(i:i+10,11,1)) + i;
    f = fft(d);
    t = lu(randn(50));
    q = ifft(f);
    v(i) = q;

    fprintf('%g, ',i);
    if mod(i,50)==0, fprintf('\n'); end
end
fprintf('\n')
```

Then run the profiler using the following code. Inspect the profiler window and use this information

to make the code run faster.

```
profile on
testfun
profile viewer
```

Answers

1. You can use the `mod` or `rem` functions to determine if a number is an integer. (Note: "integer" in math is different from "integer" in computing! We care about the former, where "integer" means whole numbers.)

```
% check for integers
if mod(m,1)~=0 || mod(n,1)~=0
   error('Error! m and n must be integers.')
end

% check for range
if (m<1 || m>30) || (n<1 || n>30)
   error('Error! m and n must be between 1 and 30.')
end

s = rand;
% check s
while s<.1
   s = rand; % pick a new s if too small
end
```

2. One option:

```
function m = mymean(x)
% function m = mymean(x);
%   Computes the arithmetic mean of numbers in input vector.
%
%     INPUTS:  x - vector of numbers
%     OUTPUTS: m - average value
%
% function written by mikexcohen.com

%% input checks
if ~isnumeric(x) % check for number data
   error('Error! Input must be numeric.')
end

if sum(size(x)==1)~=1
   error('Error! Input must be a vector.')
end

if length(x)<5
   error('Error! Input must be at least 5 elements.')
end
```

```
%% do it
m = sum(x)/length(x);
```

3. Here's the code:

```
n=6; k=3;
mod((1:n)-1,k)+1
```

4. I'll leave this one to you.

5. From inside the `linspace` function, type `assignin('base','c',c)`

6. Don't worry about about the functions `det`, `fft`, `lu`, `ifft`. You can learn more about them in my courses on Fourier analysis, signal processing, and linear algebra. But the `repmat` function is useful to know. Mostly the purpose of this exercise is to introduce you to the `profile` function.